# Modeling Workloads and Devices for IO Load Balancing in Virtualized Environments

Ajay Gulati
VMware Inc.
agulati@vmware.com

Chethan Kumar
VMware Inc.
ckumar@vmware.com

Irfan Ahmad
VMware Inc.
irfan@vmware.com

## ABSTRACT

Virtualization has been effective in providing performance isolation and proportional allocation of resources, such as CPU and memory between VMs by using automated distributed resource schedulers and VM migration. *Storage VMotion* allows users to migrate virtual hard disks from one data store to another without stopping the virtual machine. There is a dire need for an automated tool to manage storage resources more effectively by doing virtual disk placement and load balancing of workloads across multiple data stores. This problem[1] is quite challenging because it requires modeling both VM workloads and characterizing underlying devices. Furthermore, device characteristics such as number of disks backing a LUN, disk types etc. are hidden from the hosts by the virtualization layer at the array.

In this paper, we propose a storage resource scheduler (SRS) to manage virtual disk placement and automatic load balancing using Storage VMotion. SRS has three main components: 1) Model a workload using workload specific characteristics such as IO sizes, read-write ratio, number of outstanding IOs etc. 2) Model storage devices by transparently monitoring device dependent statistics such as IO latencies and 3) Suggest VM disk migrations to improve overall performance and do load balancing across devices. Our initial results lead us to believe that we can effectively model workloads and devices to improve overall storage resource utilization in practice.

## 1. INTRODUCTION

Live migration of virtual machines has been used extensively in order to manage CPU and memory, do load balancing, and improve overall utilization of host resources. Tools like VMware's Distributed Resource Scheduler (DRS) perform automated placement of virtual machines (VMs) on a cluster of hosts in an efficient and effective manner [19]. However, managing IO resources to do better placement and load balancing has been an open problem so far. Storage systems are typically provisioned based on capacity and reliability needs. Diverse IO behavior from various workloads and hot-spotting can cause significant imbalance across devices over time. Thus there is a dire need for a storage resource scheduler to automate the task of managing storage devices based on workload characteristics, application requirements and user-set policies such as reliability, availability etc.

In spite of a lot of research towards storage configuration, allocation and automatic data placement [4, 5, 7–9], most storage administrators in IT organizations rely on rules of thumb and ad hoc techniques, both for configuring a storage array and laying out data on different volumes. For example, one of the common ways to place two top-tier workloads is to create separate RAID groups on disjoint sets of disks. Over-provisioning is another technique commonly used to mitigate real or perceived performance issues. These techniques, although helpful in some cases, cannot be generalized for all. Since over-provisioning and hard-partitioning can be very expensive and inefficient, their application should be guided by workload characterization, device modeling and analysis. This problem is harder than doing CPU and memory load balancing because storage is a stateful resource and IO performance strongly depends on workload and device characteristics. We believe that existing solutions for automated IO placement and storage configuration have not been widely adopted for two main reasons: (1) lack of simple workload and device models that can be easily obtained in a live system (2) lack of a primitive to do efficient and interruption-free migration without human intervention. In virtualized environments, **Storage VMotion** technology allows users to migrate VM virtual disks from one storage device (data store) to another without interrupting the VM or any applications running inside it [20]. The usage so far has been manual, but Storage VMotion can act as the main primitive for live migration of workloads with zero down-time.

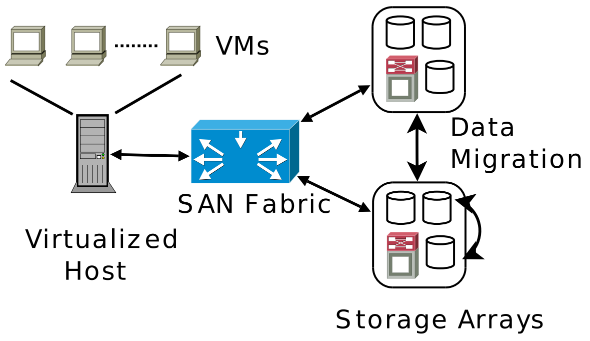


**Figure 1: Data migration using Storage VMotion**

Figure 1 shows a setup with one host having access to multiple data stores, which is quite common for virtualized data centers. The storage array is carved up into groups of disks with some RAID level configuration, also called as RAID-groups. Each such RAID-group is further divided into logical units of storage (LUNs) which are exported to hosts as storage devices. The goal of storage resource management is to do workload-aware placement and migrations of virtual disks on LUNs in order to improve IO performance as well as utilization of storage devices.

[1]Although we are using virtualized environments as our primary use case, the problem and ideas proposed in the paper are quite general and applicable to other scenarios as well.

Towards this goal, we propose Storage Resource Scheduler (SRS), which has three main components. We first model the workloads based on some of the key application-level characteristics that can be easily collected in a live system. Then we model the underlying storage devices based on the device specific measurements, which again can be done at each host. Finally we design an analysis engine that uses data from first two steps to recommend storage vmotions in order to mitigate hot-spotting.

For characterization, we partition the measurements into two sets. First are the properties that are inherent to a workload and independent of the underlying device such as seek-distance profile, IO size, read-write ratio and number of outstanding IOs. Second are device dependent measurements such as IOPS and IO latency. We use these sets to model workloads and devices respectively. We capture the measurements about IO patterns of workloads inside the hypervisor. This is done in a lightweight manner, transparent to the VMs. Device-level measurements are also obtained at each host and sometimes aggregated across hosts if the LUN is shared using a clustered file system. Based on these measurements, we assign a single load metric $\mathscr{L}$ to each workload and a single performance metric $\mathscr{P}$ to each LUN. Finally, the analyzer tries to assign the load in proportion to the performance of each storage device.

In the rest of the paper, we first present details of our workload characterization and modeling techniques in Section 2. Section 3 presents the modeling techniques to compute the performance metric of each device followed by an outline of the load balancing engine in Section 4. Section 5 presents the results of our preliminary evaluation. Section 6 provides a survey of relevant prior work and finally we conclude with some directions for future work and open problems in Section 7.

## 2. WORKLOAD CHARACTERIZATION

Any attempt at intelligent IO-aware layout policies must start with storage workload characterization as an essential first step. For each workload we track the following parameters: *seek distance, IO sizes, read-write ratio and average number of outstanding IOs*. In VMware ESX Server hypervisor, these parameters can be easily obtained for each VM and each virtual disk using a utility called *vscsiStats* [3]. The utility is online, very light-weight and transparent to the VMs running on top of the hypervisor. It generates histograms as well as averages for the workload parameters mentioned above. All of the data points are maintained for both reads and writes to clearly show any anomaly in the application or device behavior towards different request types.

We believe that the four measured parameters (i.e. randomness, IO size, read-write ratio and average outstanding IOs) are inherent to a workload and are mostly independent of the underlying device. The device dependent characteristics are the actual IO latencies and throughput observed for a workload, which we are not considering for the workload modeling. The model tries to predict the load that a workload might induce on a storage device. This load can be measured and evaluated in many ways. We are using the IO latency seen by a workload as a metric to estimate the load induced by a workload and to validate our model. Using IO latencies for validation creates some dependence on the underlying device and storage array architectures. So some inaccuracies in modeling can be attributed to this particular way of validating the model.

In order to do the modeling we ran 750 configurations varying the values of each of these parameters, using Iometer [1] inside a Microsoft Windows 2003 VM accessing a 4 disk RAID-0 LUN on a EMC CLARiiON array. The set of values chosen are a cross-product of:

|  |  |
|---|---|
| Outstanding IOs | {4, 8, 16, 32, 64} |
| IO size (in KB) | {8, 16, 32, 128, 256, 512} |
| Read% | {0, 25, 50, 75, 100} |
| Random% | {0, 25, 50, 75, 100} |

For each of the configurations we obtain the values of average IO latency and IOPS, both for reads and writes. For workload modeling, we looked at the variation of average IO latency for each one of these parameters keeping others fixed. Figure 2(a) shows the relationship between outstanding IOs and IO latency for various workload configurations. We note that latency varies linearly with the number of outstanding IOs (denoted as OIOs) for all the configurations. This is expected because as the total number of OIOs increase the overall queuing delay would increase linearly with it. For very small number of OIOs, we may see non-linear behavior because of the improvement in device throughput but over a reasonable range (8-64) of OIOs we do see very linear behavior. Similarly, IO latency varies linearly with the variation in IO sizes as shown in Figure 2(b). This is because the transmission delay increases linearly with IO size.

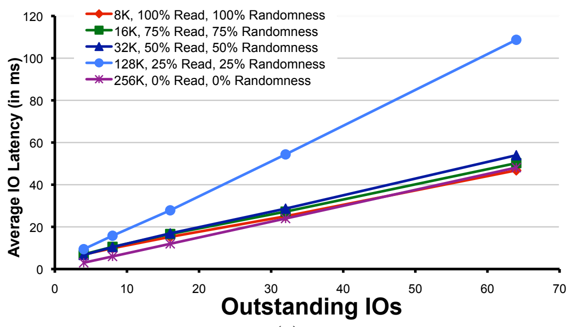
Figure 2(c) shows the variation in IO latency as we increase the percentage of reads in the workload. Interestingly, the latency again varies linearly with read percentage except for some non-linearity around corner cases such as completely sequential workload. We used the percentage of read as a variable because, for most cases, we noticed that the read latencies were almost an order of magnitude higher than the write latencies. This is mainly due to the fact that writes return once they are written to the cache at the array and the latency of destaging is hidden from the application. In cases, where the cache is completely filled, the writes may see latency closer to the reads. We believe that to be less common case especially given the burstiness of most enterprise applications [12]. Finally we looked at the variation of latency with random% (shown in Figure 2(d)) and noticed that it is linear with a very small slope, except for a big drop in latency for the completely sequential workload. These results show that except for extreme cases such as 100% sequential or 100% write workloads, the behavior of latency with respect to these parameters is quite close to linear[2].

Based on these observations, we modeled the IO latency ($L$) of a workload using the following equation:

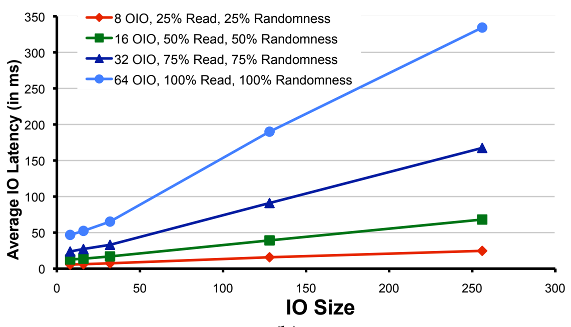
$$L = \frac{(K_1 + OIO)(K_2 + IOsize)(K_3 + \frac{read\%}{100})(K_4 + \frac{random\%}{100})}{K_5} \quad (1)$$

We compute all of the constants in the above equation using the data points available to us. We explain the computation of $K_1$ here, other constants $K_2, K_3$ and $K_4$ are computed in a similar manner. To compute $K_1$, take two latency measurements with different OIO values and same value for other three parameters. Then by dividing

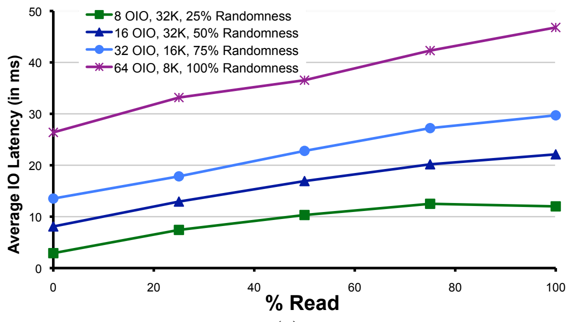
[2]The small negative slope in some cases in Figure 2 with large OIOs is due to prefetching issues in our target array firmware. This effect went away when prefetching is turned off.



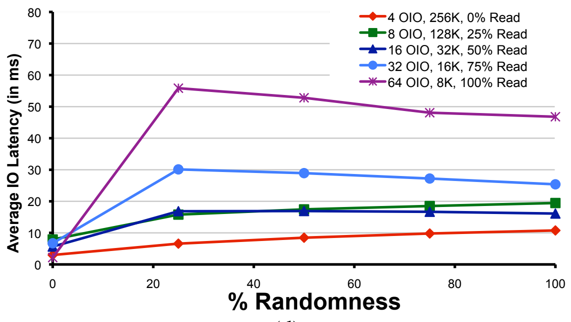
(a)



(b)



(c)



(d)

**Figure 2: Variation of IO latency with respect to each of the four workload characteristics: outstanding IOs, IO size, % Reads and % Randomness.**

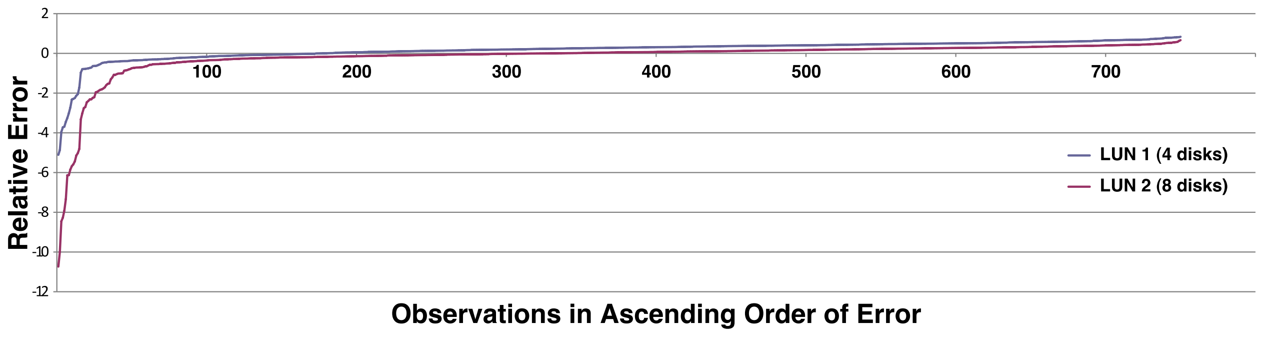


**Figure 3: Relative error in latency computation based on our formula and actual latency values observed.**

the two equations we get:

$$\frac{L_1}{L_2} = \frac{K_1 + OIO_1}{K_1 + OIO_2} \tag{2}$$

$$K_1 = \frac{OIO_1 - OIO_2 * L_1/L_2}{L_1/L_2 - 1} \tag{3}$$

We compute the value of $K_1$ for all pairs where the three parameters except OIO, are same and take the median of the set of values obtained as $K_1$. The values of $K_1$ are in a range, with some outliers and picking a median ensures that we are not biased by few extreme values. We repeat the same procedure to obtain other constants in the numerator of equation (1). To get the value of $K_5$, we compute a linear fit between actual latency values and the value of numerator based on $K_i$ values. Linear fitting returns the value of $K_5$ that minimizes the least square error between the actual measured values of latency and our estimated values. Once we determined all the constants of the model in Equation 1, we compared the two latency values. Figure 3 (LUN 1) shows the relative error between the actual and computed latency values for all workload configurations. Note that the computed values do a fairly good job of tracking the actual values, suggesting that our modeling equation is good at predicting latency based on workload parameters.

In order to validate our modeling technique, we ran the same identical 750 workload configurations on a different LUN on the same EMC storage array with 8 disks. We used the same values of $K_1$, $K_2, K_3$ and $K_4$ as computed before. Since the disk types and RAID configuration was identical, $K_5$ should vary in proportion with the number of disks, so we doubled the value, as the number of disks is doubled in this case. Figure 3 (LUN 2) again shows the actual and computed latency values for various workload configurations. Note that the computed values based on the previous constants again do a fairly good job of tracking the actual values. We noticed that most of the error values are due to the poor prediction for corner cases

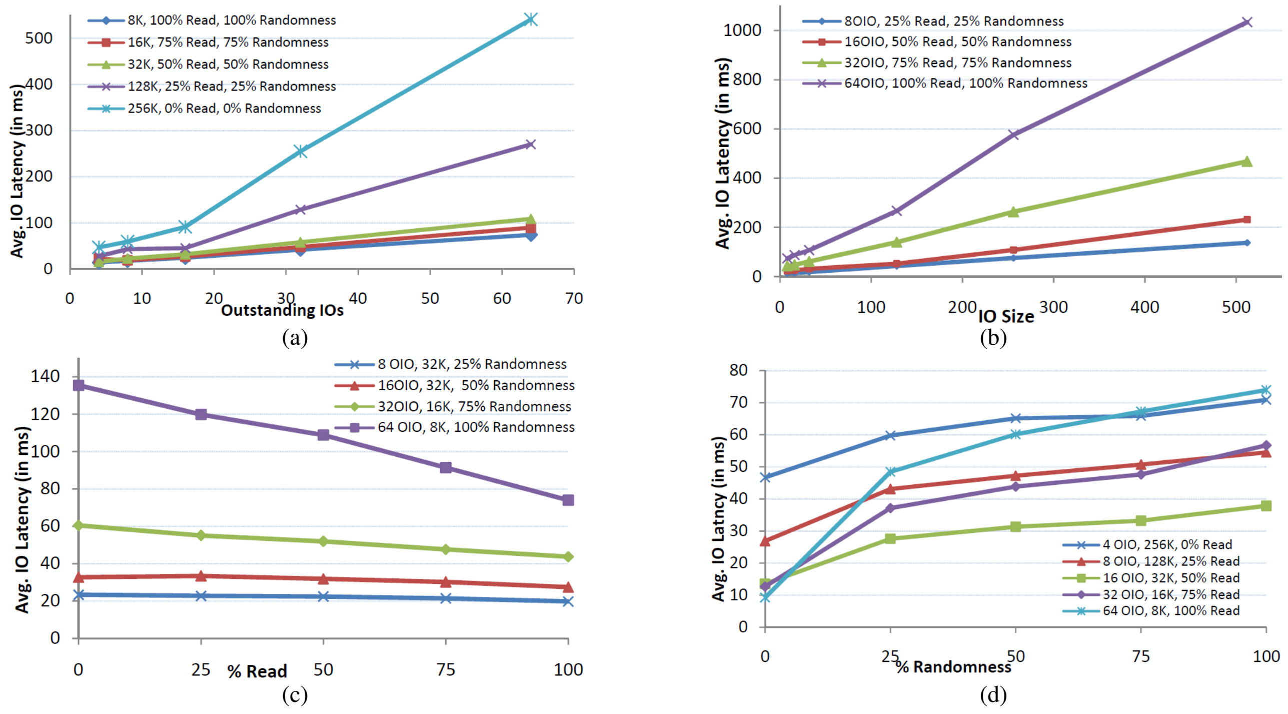


**Figure 4: Variation of IO latency with respect to each of the four workload characteristics: outstanding IOs, IO size, % Reads and % Randomness.**

such as 100% sequential, 100% writes etc.

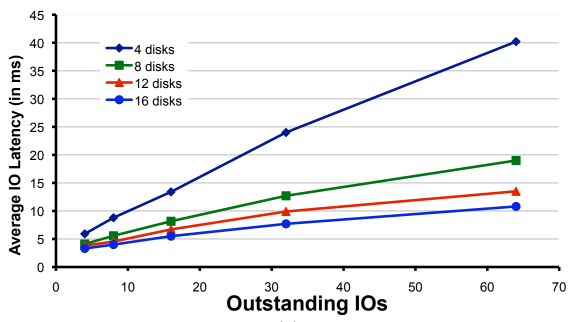
To understand variation across different storage architectures, we ran similar 750 tests on a NetApp FAS-3140 storage array. The experiments were run on a 256 GB VM disk created on a 500 GB LUN backed up by a 7 disk double parity RAID group. Figures 4(a), (b), (c) and (d) show the relationship between average IO latency with OIOs, IO size, Read% and Random% respectively. Again for OIOs, IO size and Random% we observed a linear behavior with positive slope, but for Read% case, the slope was close to zero or slightly negative. We also noticed that the read latencies were very close or slightly smaller than write latencies in most cases. We think this is due to a small NVRAM cache in the array (512 MB). The writes are getting flushed to the disks in a synchronous manner and array is giving slight preference to the reads over writes. We again modeled the system using Equation 1 and computed the relative error in the measured and computed latencies, based on the $K_i$ values computed using the NetApp measurements. Figure 5 shows the relative error for all 750 cases. We looked into the mapping of cases with high error with the actual configurations and noticed that 99% of those configurations are completely sequential workloads. This shows that our linear model over-predicts the latency for 100% sequential workloads because the linearity assumption doesn't hold in such extreme cases. Figure 2(d) and 4(d) also show a big drop in latency as we go from 25% random to 0% random case. We also looked at the relationship between IO latency and workload parameters for such extreme cases. Figure 6 shows that for sequential cases the relationship between read% and IO latency is not quite linear.

In practice, we think such cases are rare and poor prediction for such cases is not as critical. In earlier work, we have done workload characterization studies using five enterprise workloads [12]: DVDStore, Microsoft Exchange, OLTP, a TPC-C like workload and a Decision Support System (DSS) benchmark. All workloads except for decision support system showed random access patterns. Readers are encouraged to look at that study [12] for more details. In summary, there are two main challenges that we are looking into as part of our ongoing effort: (1) Can we use a fixed set of constants $K_1$, $K_2$, $K_3$ and $K_4$ for all workloads? (2) how well the modeling works for dynamic workloads where the workload characteristics change over time? Is using a certain percentile value for these variables good enough for effective modeling? As future work, we plan to study the online computation of these parameters for various workloads. Next, we need to figure out the actual device characteristics and use them to compute the denominator in Equation 1.

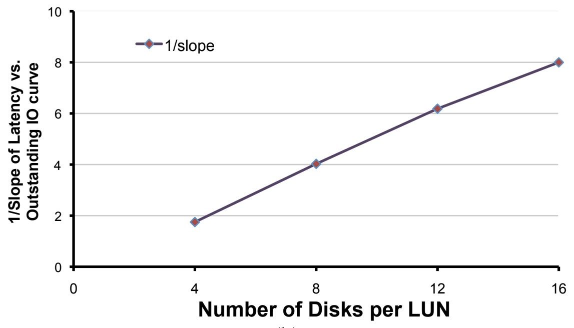
## 3. STORAGE DEVICE MODELING

So far we have talked about modeling of workloads based on the parameters that are inherent to a workload. In this section we present our device modeling technique using the measurements affected by the performance of the device. Most of the device-level characteristics such as number of disk spindles backing a LUN, disk-level features such as RPM, average seek delay, etc. are hidden from the hosts. Storage arrays only expose a LUN as a device This makes it very hard to make load balancing decisions because one needs to know if the workload is being moved from a LUN with 20 disks to a LUN with 5 disks or from a LUN with high performance drives to a LUN with slower disk drives.

For device modeling, instead of trying to obtain a white-box model of the array LUNs, we use IO latency as the main performance metric. We collect information pairs consisting of number of outstanding IOs and average IO latency observed. In a time interval,



(a)



(b)

**Figure 7: Device Modeling: different number of disks**

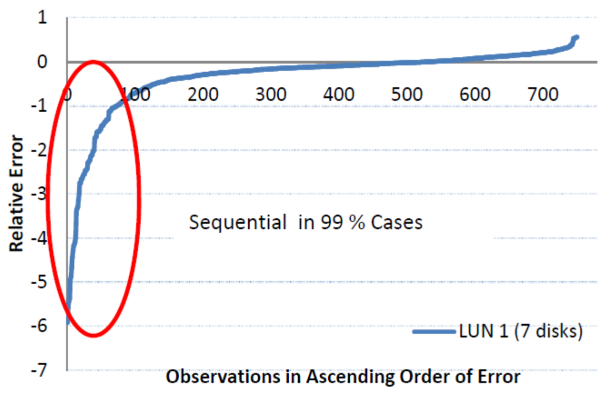


**Figure 5: Relative error in latency computation based on our formula and actual latency values observed.**

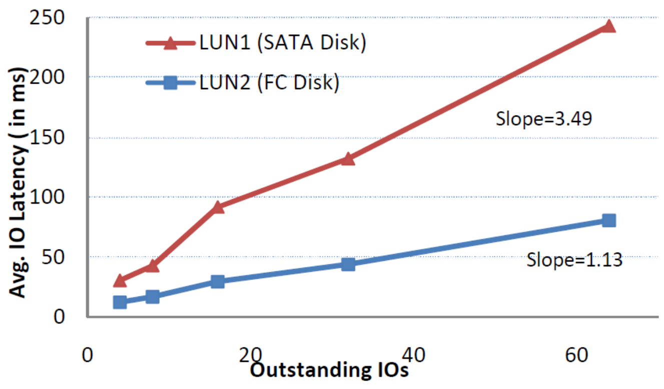


**Figure 8: Device Modeling: different disk types**

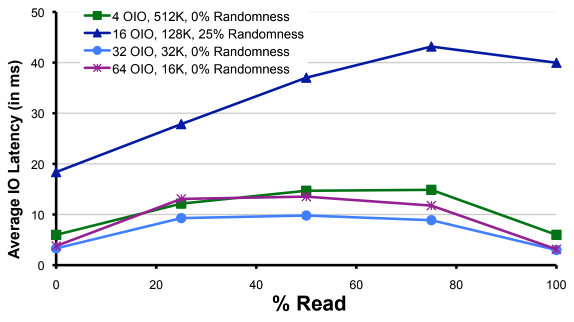


**Figure 6: Varying Read% for the Anomalous Workloads**

hosts know the average number of outstanding IOs that are sent to a LUN and they also measure the average IO latency observed by the IOs. Thus this information can be gathered in a very transparent manner without any extra overhead. For clustered environments, where multiple hosts may access the same LUN, one can aggregate this information across hosts to get aggregate results.

We have observed that IO latency increases linearly with the increase in number of outstanding IOs (i.e. load) on the array. This is also shown in earlier studies [10,11]. Given this knowledge, we use the set of data points of the form $\langle OIO, Latency \rangle$ over a period of time and compute a linear fit line which minimizes the least squares error for the data points. The slope of the the linear fit line would indicate the overall performance capability of the LUN. We believe that this should also cover cases where LUNs have different number of disks and also the ones where disks have diverse characteristics i.e. enterprise level FC vs SATA disks.

We did a simple experiment using LUNs with different number of disks and measured the slope of the linear fit line. A typical workload of 8KB random IOs is run on each of the LUNs using a Windows VM running Iometer [1]. Figure 7(a) shows the variation of IO latency with OIOs for LUNs with 4 to 16 disks. Note that the slopes vary inversely with the number of disks. Figure 7(b) shows the relationship between $1/slope$ for various LUNs with the number of disks in the LUN.

To understand the behavior in presence of different disk types, we ran an experiment on a NetApp FAS-3140 storage array using two LUNs, each with seven disks and dual parity RAID. LUN1 consisted of enterprise class FC disks (134 GB each) and LUN2 consisted of slower SATA disks (414 GB each). We created an equal sized virtual disk of size 256 GB on each of the LUNs and ran a workload with 80% reads, 70% randomness and 16KB IOs, with different values of OIOs. The workloads were generated using Iometer [1] inside a Windows 2003 VM. Figure 8 shows the average latency observed for these two LUNs with respect to OIOs. Note that the slope for LUN1 with faster disks is 1.13, which is lower compared to the slope of 3.5 for LUN2 with slower disks.

These results shows that the performance of a LUN can be estimated by looking at the slope of relationship between average latency and outstanding IOs over a long term and using that for cost benefit analysis needed for load balancing. Based on these results, we define a performance parameter $\mathscr{P}$ to be the inverse of the slope obtained by doing a linear fit on the $\langle OIO, Latency\rangle$ data pairs collected for that LUN.

## 4. LOAD BALANCE ENGINE

Load balancing requires a metric to balance over multiple resources. We use $\mathscr{L}_i$ as the numerator of Equation 1 for each workload $W_i$ as the main metric for load balancing. Furthermore, we also need to consider LUN performance while doing load balancing. We use parameter $\mathscr{P}_j$ to represent the performance of device $D_j$. Intuitively we want to make the load proportional to the performance of each device. So the problem reduces to equalizing the ratio of sum of workload metrics and LUN performance metric for each LUN. Mathematically, we want to equalize:

$$\frac{\sum_{\forall\, W_i \text{ on } D_j} \mathscr{L}_i}{\mathscr{P}_j} \tag{4}$$

The algorithm first computes the sum of workload metrics. Let $N$ be the normalized load on a device defined as:

$$N_j = \frac{\sum \mathscr{L}_i}{\mathscr{P}_j} \tag{5}$$

and $\sigma(N)$ be the variance of the normalized load. Algorithm 1 presents the pseudo-code for the load balancing algorithm. In a loop, till we get the variance the normalized load across devices under a threshold, we pick the devices with minimum and maximum normalized load to do pair-wise load migration such that both devices approach the overall average of $N$. Each iteration of the loop tries to find the set of virtual disks that need to be moved from the device with maximum ratio to the one with the minimum ratio. Optimizations in terms of data movement are mentioned later in this section. Doing a perfect balancing between these two devices is a variant of subset-sum problem which is known to be NP-complete. We are planning to use one of the approximations [14] proposed for this problem with a quite good competitive ratio of 3/4 with respect to optimal. We plan to test multiple heuristics and various ways of doing the selection of which workload to move from a higher loaded device to another, as explained next.

**Algorithm 1**: Load Balancing Step

```
foreach device j do
    foreach workload i currently placed on device j do
        $S += \mathscr{L}_i$
    $N_j \longleftarrow S/\mathscr{P}_j$
while $\sigma(N) < varianceThreshold$ do
    $D_1 \longleftarrow$ Device with maximum normalized load
    $D_2 \longleftarrow$ Device with minimum normalized load
    $N_1, N_2 \longleftarrow$ PairWiseRecommendMigration($D_1$, $D_2$)
```

**Workload/Virtual disk Selection:** We can bias the search function in many ways: (1) pick virtual disks with the highest value of $\mathscr{L}_i/(disk\ size)$ first, so that the change in load per GB of data movement is higher leading to smaller data movement (2) pick virtual disks with smallest current IOPS/$\mathscr{L}_i$ first, so that the impact of movement right now is minimal. Other constraints such as affinity between virtual disks and data stores, access to the destination data store at the current host running the VM, etc. can be handled as part of virtual disk selection in this step. Overall, this step incorporates any cost-benefit analysis that is needed to chose which VMs to migrate in order to do load balancing. After computing these recommendations, they can either be presented to the user as suggestions or can be carried out automatically during periods of low activity.

**Initial Placement:** Better decision in terms of initial placement of workloads or virtual disks is as important as future migrations. Initial placement gives us a good way to reduce potential imbalance issues in future. In this design, we plan to use the overall normalized load (i.e. ratio of workload metrics and LUN performance metric) as an indicator of current load at the LUN. After resolving hard constraints in terms of reliability or types of storage, we choose the LUN with the minimum value of the normalized load as defined above. This ensures that with each initial placement, we are naturally reducing the overall standard deviation among normalized load on LUNs.

### 4.1 Discussion

In one of our previous studies [12] we studied the impact of consolidation on various kinds of workloads. We observed that when random workloads and the underlying devices are consolidated, they tend to perform at least as good or better in terms of handling bursts and the overall impact of interference is very small. Although when we place random and sequential workloads together, we saw degradation in throughput of sequential workloads. We also studied two top-tier enterprise applications: Microsoft Exchange and Oracle database (Swingbench and TPC-C like workloads) and found all of them to have random access patterns. So in the common case we don't envision too many sequential workloads in use. However, to handle specific workloads such as log virtual disks, decision support systems, multi-media servers, we plan to incorporate two optimizations: first isolating them on separate set of spindles to reduce interference and second allocating fewer disks to the sequential workloads because their performance is less dependent on the number of disks as compared to random workloads. This can be done be setting soft affinity of these workloads to specific LUNs and setting anti-affinity of such workloads with random ones. Thus we can bias our greedy load balancing heuristic to consider such affinity rules while making placement decisions.

Whereas we consider these optimizations as part of our future work, we believe that the proposed techniques are useful for a wide variety of cases even in the current form since existing systems do not provide effective support for IO load balancing. In some cases users may have reliability or other policy constraints such as RAID-level, mirroring, etc., attached to VM disks. In those cases a set of devices would be unsuitable for some VMs and we would treat that as a hard constraint in our load balancing mechanism while recommending placements and migrations.

## 5. EXPERIMENTAL EVALUATION

In this section we discuss some of the preliminary results of our experiments with the modeling metrics and load balancing mechanism discussed earlier. For our experiments we use five workloads with very different characteristics, as shown in Table 1. All workloads are generated using Iometer running inside a Windows 2003 VM. The workloads are represented as W1, W2, W3, W4 and W5. We also used two data stores: Datastore1 ($D_1$) and Datastore2 ($D_2$) backed by 5 and 10 disks RAID groups with identical disks on an EMC CLARiiON array.

| | Before Running SRS | | | After Running SRS | | |
|---|---|---|---|---|---|---|
| Workers | Latency (ms) | Throughput (IOps) | Location | Latency (ms) | Throughput (IOps) | Location |
| W1 | 19 | 800 | $D_1$ | 17 | 881 | $D_2$ |
| W2 | 13 | 650 | $D_2$ | 18 | 390 | $D_2$ |
| W3 | 18 | 100 | $D_2$ | 26 | 70 | $D_2$ |
| W4 | 1 | 4000 | $D_2$ | 1.7 | 3600 | $D_2$ |
| W5 | 20 | 760 | $D_1$ | 13 | 1200 | $D_1$ |

**Table 2: Workload Migration Recommendation by SRS. Average latency and IO Throughput for workloads $W1$ through $W2$ before and after migration.**

| | | | Before Running SRS | | | After Running SRS | | |
|---|---|---|---|---|---|---|---|---|
| Datastores | # Disks | $K_5$ | Latency | IOPs | Total Load ($\mathscr{L}$) | Latency | IOPs | Total Load ($\mathscr{L}$) |
| $D_1$ | 5 | 1 | 19 | 1560 | 10472 | 13 | 1200 | 6314 |
| $D_2$ | 10 | 2 | 3 | 4750 | 3854 | 6.5 | 4800 | 5933 |

**Table 3: Results of Workload Migration Recommendation by SRS. Latency, IOPS and overall load on Datastores $D_1$ and $D_2$.**

| Workers | OIO | IOSize (KB) | Random % | Read % |
|---|---|---|---|---|
| W1 | 16 | 4 | 100 | 75 |
| W2 | 8 | 64 | 10 | 100 |
| W3 | 2 | 128 | 90 | 50 |
| W4 | 4 | 32 | 0 | 0 |
| W5 | 16 | 32 | 100 | 75 |

**Table 1: Details of Test Workloads.**

We first ran the base case where W1 and W5 are located on Datastore2 and others are located on Datastore1. Table 3 shows the overall latency and IOPS obtained by two datastores. The last column represents the sum of load metric for the two datastores based on workloads running on each of them. Note that the overall load and also the average latency is very different for the two datastores. Table 2 shows the average latency and IOPS for each of the individual workloads for this case. Again we notice that workloads on Datastore2 see much lower latency as compared of workloads on Datstore1.

Then we computed the load metric for each workload using numerator in Equation 1 and did workload migration to equate the load on two data stores based on overall workload and device model. In this case we assume that device model has been computed separately and we know that one device is twice as powerful as the other. This is reflected in the $K_5$ for each device mentioned in Table 3. Based on the recommended migration computed by our algorithm, we moved W1 to Datastore2. Table 3 shows the overall latency and IOPS obtained by two datastores after the movement. Note that average latency has moved from $\langle 19, 3\rangle$ ms to $\langle 13, 6\rangle$ ms which is much more balanced compared to the first case. In this case this turns out to be the optimal assignment as well based on the granularity of the migration. Table 2 presents the individual workload measurements for this case. Again notice that W5 gets much lower latency as it is running alone on Datastore1. Some workloads see an increase in latency but that is expected when we move workloads to an under-utilized datastore. The sum of IOPS may be different because workloads don't always add up when they are run together on a datastore. Although we did notice that both OLTP workloads got higher number of IOPS and the sequential workload got hit by the move.

In general we expect the IO throughput to not decrease substantially except for the cases when highly sequential workload streams are hit by the interference from other workloads. We plan to detect such cases and avoid placing such workloads with others as much as possible.

## 6. RELATED WORK

There has been a lot of prior work related to storage configuration, workload characterization and placement. Hippodrome [5] automates storage system configuration by iterating over three stages: analyze workload, design system and implement design. Similarly Minerva [4] uses a declarative specification of application requirements and device capabilities to solve a constraint based optimization problem for storage system design. These tools are trying to solve a more difficult problem of optimizing overall storage system design. Our model is designed to load balance an existing storage system. Overall we believe that such tools are complimentary to our work and would work better with the knowledge gained through our workload characterizations. Other studies have looked into file system access patterns [2, 13, 17] and IO patterns at disks [15], mostly in context of user workloads in an organization. Although these are quite useful for file system level analysis, we mainly focus on VM workloads running enterprise and other applications.

Studying RAID performance for different workloads has been an active area of research. Chen et. al. [6–8] have shown that stripe size, IO sizes and RAID level can have a wide impact on overall performance. They also showed that applications with large accesses do better with RAID 5 and application using large number of smaller accesses (e.g. OLTP), perform better using mirrored RAID. Researchers have also tried to model disk drives [16] and storage arrays [9, 18] to automate the analysis and prediction of workload behaviors. This is especially hard given the device characteristics of disks and all the complexity built into an array in terms of service processors, buses, controller caches etc. Most existing commercial products such as EMC CLARiiON, HP SureStore, IBM Tivoli, Equalogic PS series provide a layer of abstraction for storage devices and management tools to partition and configure devices as needed. Some of them also do automatic load balancing by moving data within or across arrays. However, configuring storage arrays using these tools is still a difficult task. We try to model workloads and devices based on simple statistics that can be collected transparently and efficiently inside a hypervisor. We believe that our models are simple yet effective for use in practice.

## 7. CONCLUSIONS AND FUTURE WORK

In this paper, we presented modeling techniques for workloads and storage devices in order to do load balancing of workloads over a set of devices. The workloads were modeled independent of underlying device using the parameters inherent to a workload such as seek distances, read-write ratio, average IO size and outstanding IOs. For device modeling we used statistics such as IO latency with respect to outstanding IOs which is dependent on the storage device. We also presented a load balancing engine that can do migrations in order to balance overall load on devices in proportion to their capabilities. Our preliminary evaluation shows that the proposed modeling works well in characterizing workloads and devices.

Some of the research problems that have emerged from this work are: (1) How to characterize dynamic workloads for load balancing? Are percentile values for workload parameters good enough in real systems? (2) Can we use static values of constants such as $K_1$ to $K_4$ for workloads running on different devices or do we need online estimation? (3) How to measure and predict interference among various workloads accessing a device? (4) How to suggest storage configuration changes to an administrator based on online workload monitoring. Ultimately the task of workload monitoring, device modeling and load balancing needs to happen in a feedback loop over time to handle churn in today's storage environments.

**Acknowledgements:** Thanks to Carl Waldspurger, Minwen Ji and our shepherd Arif Merchant for valuable discussions and feedback. Thanks also to Karan Kumar for help with running some of the experiments and gathering data.

## 8. REFERENCES

[1] Iometer. http://www.iometer.org.

[2] N. Agrawal, W. J. Bolosky, J. R. Douceur, and J. R. Lorch. A five-year study of file-system metadata. *Proceedings of the 5th Conference on File and Storage Technologies (FAST '07)*, Feb 2007.

[3] I. Ahmad. Easy and Efficient Disk I/O Workload Characterization in VMware ESX Server. *IISWC*, Sept. 2007.

[4] G. A. Alvarez and et al. Minerva: an automated resource provisioning tool for large-scale storage systems. In *ACM Transactions on Computer Systems*, pages 483–518, November 2001.

[5] E. Anderson and et al. Hippodrome: running circles around storage administration. In *Proc. of Conf. on File and Storage Technology (FAST'02)*, pages 175–188, January 2002.

[6] P. M. Chen, G. A. Gibson, R. H. Katz, and D. A. Patterson. An evaluation of redundant arrays of disks using an amdahl 5890. *SIGMETRICS Perform. Eval. Rev.*, 18(1):74–85, 1990.

[7] P. M. Chen and E. K. Lee. Striping in a raid level 5 disk array. In *SIGMETRICS '95*, pages 136–145, New York, NY, USA, 1995. ACM.

[8] P. M. Chen and D. A. Patterson. Maximizing performance in a striped disk array. *SIGARCH Comput. Archit. News*, 18(3a):322–331, 1990.

[9] T. Denehy, J. Bent, F. Popovici, A. Arpaci-Dusseau, and R. Arpaci-Dusseau. Deconstructing storage arrays, 2004.

[10] A. Gulati and I. Ahmad. Towards distributed storage resource management using flow control. In *International Workshop on Storage and I/O Virtualization, Performance, Energy, Evaluation and Dependability (SPEED)*, February 2008.

[11] A. Gulati, I. Ahmad, and C. Waldspurger. Parda: Proportionate allocation of resources for distributed storage access. In *Usenix FAST*, February 2009.

[12] A. Gulati, C. Kumar, and I. Ahmad. Storage workload characterization and consolidation in virtualized environments. In *Workshop on Virtualization Performance: Analysis, Characterization, and Tools (VPACT)*, 2009.

[13] J. K. Ousterhout, H. D. Costa, D. Harrison, J. A. Kunze, M. Kupfer, and J. G. Thompson. A trace-driven analysis of the unix 4.2 bsd file system. *SIGOPS Oper. Syst. Rev.*, 19(5):15–24, 1985.

[14] B. Przydatek. A fast approximation algorithm for the subset-sum problem.

[15] C. Ruemmler and J. Wilkes. UNIX disk access patterns. In *Usenix Conference*, pages 405–420, Winter 1993.

[16] C. Ruemmler and J. Wilkes. An introduction to disk drive modeling. *IEEE Computer*, 27(3):17–28, 1994.

[17] M. Satyanarayanan. A study of file sizes and functional lifetimes. *SIGOPS Oper. Syst. Rev.*, 15(5):96–108, 1981.

[18] M. Uysal, G. Alvarez, and A. Merchant. A modular, analytical throughput model for modern disk arrays, August 2001.

[19] VMware, Inc. Resource Management with VMware DRS, 2006. `http://www.vmware.com/pdf/vmware_drs_wp.pdf`.

[20] VMware, Inc. VMware storage vMotion: Non-disruptive, live migration of virtual machine storage, 2007. `http://www.vmware.com/files/pdf/storage_vmotion_datasheet.pdf`.